DR1 est en adresse 5

CREATION D'IMAGE ISO

- Le disque dur (ISO) doit être configuré en adresse SCSI 5 switchs 1 et 3 sur ON

- Copier sur le Disque dur BACKUP les répertoires :

  C
  L
  DEVS
  S
  FONTS
  LIBS

(à partir du 1.3.3.)

← mettre la bibl. de Amos

Avec les commandes et fichiers du WORKBENCH 1.33

- Copier dans le Root du disque dur BACKUP le fichier CDTV.TM contenu dans la disquette ISO DEVPAC.

- Créer dans le répertoire S le fichier STARTUP-SEQUENCE suivant :

```
C:SETPATCH >NIL:
RMTM
CDTVPREFS
CD CDO:
NOM DE LANCEMENT DE VOTRE APPLICATION
~~ENDCLI >NIL:~~
RUNBACK
```

Nom Programme (rescrire Workbench)

- Copier depuis la disquette WBCONFIG le fichier SYSTEM-CONFIGURATION dans le répertoire DEVS dus disque dur BACKUP.

- Booter sur la disquette ISO DEVPAC

- Sous SHELL -> **CD RAM:ISO2**
  ISO "NOM"

"NOM" : Nom du disque contenant l'application exemple-> TOTO:

Demandes	Réponses
Volume ISO9660	**DR1:**
VolumeIdentifier	NOM DE LA SOCIETE
VolumeSetIdentifier	VOL1
PublisherIdentifier	NOM DE LA SOCIETE

- Sous SHELL -> **BYTEDRIVE SCSI.DEVICE 5**
  MOUNT DR1:

- Autre SHELL -> **BUILDTRACK CONTROLFILE -W**

Une fois l'image ISO générée, durée environ 1 heure/100 Mo

SCSIVIEW (noter le nombre de blocs et le multiplier par 2048, noter ce chiffre pour le pressage)

FIXTM répondre Y et Y

Commodore

Commodore France

152, avenue de Verdun
92137 Issy-les-Moulineaux Cedex
Tél.: (1) 46 44 55 55
Télex: 631558
Fax: (1) 46 44 59 30

SARL au Capital 8 400 000 Francs - RCS Nanterre B 331 721 522 - SIRET 331 721 522 00026 - Code Activité 5911
Siège social: 152, avenue de Verdun - 92137 Issy-les-Moulineaux Cedex

Commodore FRANCE

Premier
EDIT...
ORLANDO!??

Souhaité ☒
☐
☐

Nom: _____

Adresse: _____

Ville: _____

Pays: _____

Téléphone: (____) _____

Contacts: _____

_____

Les personnes mentionnées comme cor
confidentialité.

Nombre de personnes impliquées dans l

Rattra
mach } C:
rmtm

écrase
Syst-Cfigati
→ devs:
Syst-Cfigati BLK+P

Je certifie que ces informations sont exactes.
Je certifie que j'ai lu et pris connaissance de la
ADSPE, et j'accepte ces conditions.

J'accepte que COMMODORE FRANCE puiss
service support développeur ADSPE à tout mo
puisse suspendre ma participation au Service S

Nom: _____

Signature: _____ *Date:* _____

Numéro de développeur: _____ (Fourni par COMMODORE)

Commodore FRANCE

A.D.S.P.E.

Description de produit

Un formulaire par produit (existant ou en développement)

Date:_____ Société/Particulier :_____

Nom du produit:_____

Catégorie du produit:_____

Version:_____ Prix de vente:_____ Date de disponibilité:_____

Ce produit est/sera disponible pour:

A1000 ☐ A500 ☐ A2000 ☐ A2500 ☐ A3000 ☐

A 600 ☐ CDTV ☐

Caractéristiques:

PAL ☐ NTSC ☐ Claviers nationaux ☐

Le produit est/sera disponible dans les langues:

_____

La documentation est/sera disponible dans les langues:

_____

Description brève du produit:

Veuillez indiquer le nom de vos éditeurs/distributeurs si vous les connaissez.

Available Memory for Various CDTV Configurations

RAM space is at a premium in the CDTV. It was felt that a chart of the memory available upon booting various configurations of hardware capable of playing CDTV discs would be a useful aid to assist developers in making titles work on the maximum number of configurations.

For comparison purposes, all machines were allowed to boot far enough to open the Workbench screen (in the default size) and a CLI/Shell window. This represents a reasonable starting point for developers to estimate if their application will fit into a CDTV's memory. (To properly ascertain the amount of free memory in a particular CDTV/A500(+) at run time, use the Exec library's AvailMem() function.) These results are not guaranteed for any particular configuration; they can and will change depending upon many variables, including additional hardware, overscan or interlaced screens, etc. Use this information only as a guideline, not as a lifeline.

Important note about Workbench V1.3 and V 2.0. Under 1.3, the Workbench screen and initial CLI will always open upon booting with an AmigaDOS disc. Under 2.0, the Workbench screen is only opened once it is needed, and an initial CLI is only opened if the Startup-sequence attempts to output text. The "Avail" program first calculates the amount of RAM free and then prints out the results. Under 2.0, this would mean that the amount of memory printed by Avail would differ from the amount of memory actually free by the amount needed to create the Workbench screen and initial CLI.

Machine	NTSC/PAL	1.3/2.0	Peripherals	*	Available	In-Use	Maximum	Largest
CDTV	NTSC	1.3	internal HD floppy	c	799448	240704	1040152	797312
CDTV	NTSC	1.3	floppy	c	886688	153464	1040152	710472
CDTV	NTSC	2.0	floppy	c	840616	206936	1047552	689200
CDTV	NTSC	1.3	SCSI card	c	881248	158904	1040152	704904
CDTV	NTSC	1.3	floppy	c	878824	161328	1040152	878624
CDTV	PAL	2.0	floppy	c	807528	239992	1047520	792880
CDTV	PAL	2.0	none	c	828784	218768	1047552	824736
CDTV	PAL	1.3	floppy	c	877936	162216	1040152	710552
CDTV	PAL	1.3	internal HD floppy	c	790488	249664	1040152	788352
A500/570	NTSC	1.3	internal FD	c	466256	56976	523232	466208
				f	412728	104160	516888	412552
				t	878984	161136	1040120	466208
A500/570	PAL	1.3	internal FD	c	457296	65936	523232	457248
				f	412704	104184	516888	412064
				t	870000	170120	1040120	457248
A500+/570	PAL	2.0	internal FD	c	809744	237776	1047520	795144
A500+	PAL	2.0	internal FD	c	839688	207864	1047552	838648

* c=chip; f=fast; t=total
FD = Floppy Drive; HD = Hard Drive

CTrac Emulation System

Introduction

The *CTrac* CD-ROM emulation system, developed by ICOM Simulations, Inc., is a combination of hardware and software that emulates CD-ROM discs for the CDTV. The emulation system creates an image of a CD-ROM disc in a file on the hard disk drive of an Amiga. The system then acts as the CD-ROM drive by presenting the image to the CDTV, as if it were an actual CD-ROM disc. Since the emulation system replaces the CD-ROM drive in the CDTV, all formats and modes of CD-ROM can be emulated including CD-DA, CD+G and CD+MIDI.

This is a true emulation environment, not a simulation. The application actually runs in the memory of the CDTV. The emulator board replaces the CD-ROM mechanism. Every time the CDTV tries to access the CD-ROM, it accesses the emulator board. The board and the emulation software in turn access an ISO 9660 disc image on the Amiga's SCSI disk drive. The board slows down the seek times of the SCSI drive, as well as the data transfer rate, to accurately emulate the performance of a CD-ROM drive.

Until now, the only accurate way to test CD-ROM applications was to have the application pressed on to a CD-ROM disc and then tested in the CDTV. The process of pressing a disc to test an application under development is time consuming and costly, especially if the application does not work properly. Emulating CD-ROM applications with the *CTrac* emulation system not only eliminates the need to have discs pressed for testing but also allows accurate monitoring of the commands sent to the CD-ROM drive. We do recommend that a test disc be cut before mastering the application, however.

The *CTrac* CD-ROM emulation system hardware consists of a printed circuit board that plugs into an Amiga 2500 or Amiga 3000, and an interface cable. The cable connects the emulator board to the CD-ROM drive connector in the CDTV. The *CTrac* hardware together with the *CTrac* software running on the Amiga emulates the CD-ROM drive in the CDTV. The emulation is completely transparent to the CDTV.

CTrac Contents and Software Description

Hardware

The *CTrac* emulation hardware consists of the *CTrac* emulation printed circuit board, and an interface cable.

Software

The *CTrac* emulation software consists of the following files and directories:

Emulate
The emulation software

Builddisc
A tool used to create entire CD disc images for the *CTrac* emulation system. It combines track images, along with the proper subcode information, into one composite disc image. The final disc image is needed for the emulation software.

Buildtrack
A tool used to create ISO 9660 formatted tracks that are needed on the CD-ROM discs used in the CDTV. The output of this program is used as input for the BuildDisc program.

ISOUtil
A tool used to extract, update and list directories of files contained in ISO track images or from disc images that contain ISO tracks.

Libs
The Libs directory contains the requestor library that must be installed on the Amiga before running the emulation software.

QFS
The QFS directory contains further directories and files needed to run the Qwik File System on the Amiga. The Qwik File System was developed by CONSULTRON. The data rate and seek time requirements of the emulator exceed the abilities of the Amiga Fast File System. Therefore, the Qwik File System is provided and must be installed on the hard drive that will contain the disc images that will be used for emulation.

The Qwik File System is approximately 16 times faster than the Fast File System when performing directory operations and approximately 2 times faster when reading files greater than 36K. The Qwik File System must be installed on the hard drive that will contain the disc images to be emulated.

The Qwik File System is compatible with the AmigaDOS file requirements with the following exception:

16 character file/directory comments must be used instead of 79.

Install
A batch file that copies the emulation software and tools to the command directory of the Amiga boot drive.

The ISO DevPak diskette is also delivered with the Emulator. This diskette provides other tools necessary for the creation of the ISO 9660 image.

Hardware Requirements

Amiga 2500, Amiga 3000, or Amiga 3500, with one free expansion slot

The *CTrac* emulation software was designed to run on an Amiga 2500 or higher model, such as the Amiga 3000. If an Amiga 2500 will be used for development, the emulation software must be run in the 2500's 68030 mode. 3 Mbytes of RAM minimum are required.

Two or three hard disk drives, with one 600 Mbytes or larger

At least two hard disk drives should be used with the *CTrac* emulation software. One hard drive should utilize the Amiga Fast File System and contain all of the standard Amiga libraries and tools. The second hard drive, which should be the larger of the two, must have the Qwik File System installed. It will contain the disc images that will be emulated. If the image is too large, you may need a third drive.

CD-ROM discs may contain varying amounts of data up to a maximum of approximately 650 MB. Since the image of a CD-ROM disc is stored in a file on the hard drive of an Amiga, it follows that the hard drive used must be at least as large as the largest image that will be emulated. Moreover, it may be desireable to store the source data and track images that were used to build the disc image on the same hard drive. If this is done, it could more than double the amount of storage space required. Therefore, a 600 MB or larger drive is strongly recommended.

For example, assume your application requires 400 Mbytes of data and code. In this case you would probably need three hard disk drives, or a total capacity of 1200 MBytes.

- Drive 1, formatted under the Fast File System, contains your original source data and code. You develop and test your application on this drive.
- Drive 2 will receive the track image built by the BuildTrack utility. This image will be at least as large as your application code plus data (400 MB, in this case).
- Drive 3, which must be formatted under the Qwik File System, will receive the disc image built by the BuildDisc utility. Again, this image will be at least as large as your application code (400 MB).

One Drive May Be All You Need. You may combine both the track image and the disc image on one drive, if they will fit. If space allows, you can even build those images on your source drive, if you have formatted your source drive with the Qwik File System. Thus if your application contains only 30 Mbytes of code and data, you could develop it and test it under the emulator with a single 100 Mbyte drive. You may also use separate partitions of the same drive.

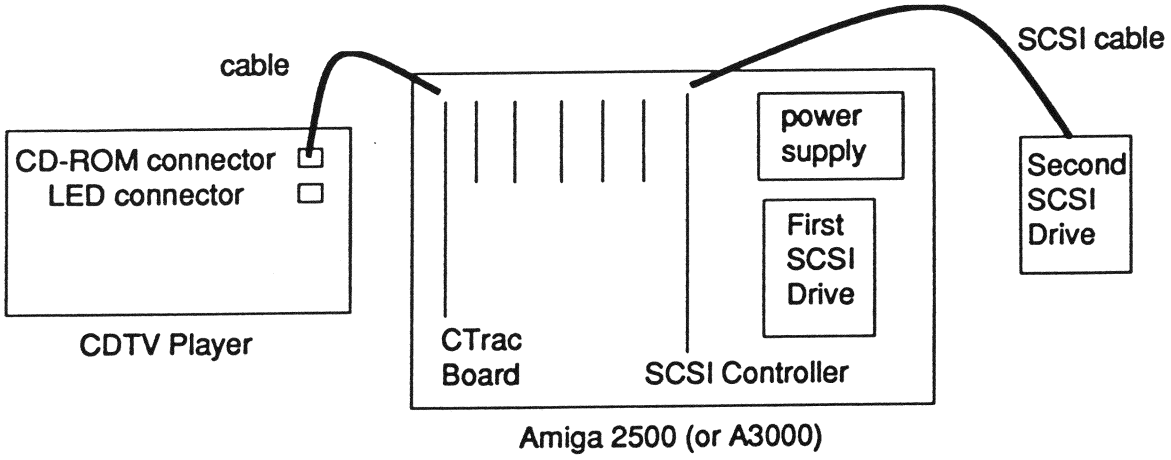
The hard drive containing the disc image must also be reasonably fast because the emulator requires at least a 175K bytes/second maintainable data rate. For example, we have tested the drive using the Quantum 40 and 100 Mbyte drives delivered with Amiga 3000s. Unfortunately, these drives are not fast enough to allow proper emulation. The emulator reports error messages indicating that it could not read data fast enough from those drives. We have successfully used Seagate 450 Mbyte drives (reference ST2502N 94241-502) to resolve these problems.

Installation of the Emulation System

Installation of the system is fairly quick, requiring approximately twenty minutes. First, the large SCSI disk must be formatted under the Qwik File System.

Next, the install program is executed to copy the tools necessary for image building and emulation to the Amiga hard disk drive.

Finally, the *CTrac* emulation board must be installed in the Amiga 2500 or 3000, and connected to the CDTV player. The figure below indicates the physical set-up of the emulation system.



Creating Disc Images and Using the Emulation Software

The following section describes the steps involved in preparing a disc image for the emulator, and using the emulator board and software to emulate that image.

For this discussion, we will assume that the application is in "mixed mode", that is, it contains a certain number of CD-DA audio tracks, as well as Amiga code and data. We will also assume the application has been written, and the data is ready. The application resides on a drive we shall name DH0:. A second hard disk, named Q0:, has been formatted under the Qwik File System to receive the image files.

Step 1

Use the iso utility to create a control file for *BuildTrack*.

The ISO utility is provided on the ISO DevPak diskette, available to licenced CDTV developers. It reads the directory structure of your AmigaDOS drive (DH0:) and creates a file which specifies which files to include in the ISO 9660 track, and the order of those files. The iso utility allows files to be sorted by size, alphabetically by name, or to remain in the same order as they were on the source disk. Iso creates an ASCII text file, which the developer can modify for further optimization.

Step 2

Use *BuildTrack* to create a track image.

The *BuildTrack* utility reads in the controlfile created by the iso utility. It verifies for ISO 9660 compatibility (maximum levels of nesting for subdirectories, filename conventions, etc.) It then generates an ISO 9660 track image on the target drive (Q0:)

Step 3

Use the *FixTM* utility

The *FixTM* utility, also on the ISO DevPak diskette, modifies the ISO image to allow for direct booting on CDTV.

Step 4

Create the *BuildDisc* controlfile.

Like the *BuildTrack* utility, *BuildDisc* is also directed by a control file. The control file, a standard text file, specifies which tracks will be included in the disc image, and in what order. It also allows other attributes of the disc to be specified, such as the length of the table of contents (TOC).

Step 5

Use *BuildDisc* to create the disc image.

BuildDisc reads the controlfile, and generates a disc image with the track images that were specified.

Step 6

Run the emulator using the disc image.

Now we can begin the actual emulation. The emulator software runs in the memory of the Amiga 2500 or 3000. It requests the name of the disc image file which should be emulated, and then begins to work.

Step 7

Reboot the CDTV.

Finally, we power on (or reset) the CDTV. The emulator software (along with the emulator board) will intercept all attempts by the CDTV to access the CD-ROM drive. They access the disc image on the SCSI drive instead. Furthermore, the board and software assure emulation of the data transfer rate and seek times of the CD-ROM mechanism.

The CDTV application can be viewed on a monitor attached to one of the CDTV video outputs. Meanwhile, the emulator software displays activity messages in its Activity Window. All accesses to the ROM are indicated, as well as all commands received by the emulator.

A Status window at the bottom of the screen displays the name of the image file being emulated, any emulator errors that have occurred during emulation, and the current state of the drive status bits (ready, audio, done, error, SpinUp, DiscIn, InfErr).

Optionally, time emulation can be suspended. If this is done, the emulator no longer emulates spin up, spin down, and seek delays. This permits testing applications at the fastest possible speed, but the emulation is no longer accurate.

Follow The Script. It is possible (and even advisable) to combine all these steps into a single script file. You can launch the script, let it run as long as needed, and return to test your emulation.

Emulator Limitations

Seek Times

The emulator approximates the seek times and other delays of the CD-ROM drive in the CDTV. Due to the variances in drives and discs, it is possible that a seek time on the emulator could be slightly different from that on the actual drive.

Tasks

In order to obtain optimal results, it is not advised to run other tasks on the Amiga 2500 or 3000 simultaneously with the emulator software. The emulator needs a large amount of CPU time, and its priority is set to 100. Furthermore, contention may develop between two tasks trying to use the SCSI bus simultaneously.

Disk space and speed requirements

As discussed above, the *CTrac* system requires large amounts of SCSI disk space—three times the size of your application. Furthermore, the disk must be a fast one, if you want to accurately emulate the data transfer rate used by the CD ROM mechanism of the CDTV player.

Speeding Up Your Titles

The Importance Of Speed

The demands of multimedia titles are different from the demands of traditional computer programs. Computer users are familiar with the idea of waiting for data to be loaded, particularly when working with floppy disks as the storage medium, but users of multimedia applications approach them in the same way that they watch a videotape of a film. Long delays with no apparent activity will soon make the users hostile to the title. They are not interested in *why* it takes ten seconds to load an animation—they want the show to go on.

Some thought must therefore be given to convincing users that the title is doing something sensible, or they will begin to stab at the buttons on their infrared controller or keyboard.

Usually, and understandably, these long waiting periods result from programmers applying quite proper traditional wisdom to a medium whose physics and demands are a little different.

Subjective Speed

A common delay on a CDTV title occurs when the user presses a button to go on to the next picture or screen in a series. Typically the title will wait until a button is pressed, and then begin loading the data. This will take *at least* one second per 150 KBytes of data, plus any seek times involved in the laser moving to the position of the data in the CD-ROM track.

If, however, that picture is loaded into a second screen buffer while the user is looking at the previous picture, it can be made to appear on the screen instantly when the user presses the button. The load time for the data has not changed, but the user's subjective impression is that the system is very responsive.

This simple technique can change a sluggish title into a snappy title with no low-level optimization of data transfer rates, but it does require planning at the design stage to make the double buffering possible. Most approaches to improving performance depend on good design at the planning stage and cannot simply be *bolted on* afterwards.

The Startup Sequence

The first impression a user has of a title is what happens when the disc is inserted. The following sequence of events takes place:

1. The system detects a new CD and resets.
2. The system examines the disc to determine if it is a CD-ROM disc or a CD-DA disc.
3. If it is CD-DA, the CD player screen is launched from ROM.

4. If it is a CD-ROM disc or mixed-mode disc (and there is no floppy disk drive with bootable disk attached), the system tries to boot the disc.
5. If there is an ISO-9660 image in track one, this is examined. If it is a CDTV CD-ROM and it has the correct copyright data in place it is booted. If the copyright is not correct, the system turns the screen red.
6. At this stage the startup sequence in the S directory of track one is executed.

It is clear that this process takes several seconds to perform before the startup-sequence itself is activated. So that the title itself can begin quickly, some thought needs to be given to the startup-sequence.

It is not normally necessary on CDTV to use an entire Workbench startup-sequence. Assuming no *Assign* statements are needed the minimum startup-sequence may be:

```
rmtm     ; remove trade-mark
MyTitle  ; executable for my title
```

A more typical Workbench 1.3 startup will need at least the following components (in no particular order):

```
rmtm                ; remove trade-mark
setpatch >NIL: r     ; patch layers etc
FF >NIL: -0         ; speed up fonts
Assign MyVol: CD0:  ; assign logical volume that was hard disk
                    ; to CD
bookit bv           ; blank screen, centered Views
MyTitle             ; run the title!
```

With these actions in the order above, the *rmtm* program will delay a few seconds and fade the copyright notice. The screen will (if the preferences set in devs:system-configuration are black screen *and they most likely should*), then be black. The CDFS then finds and launches *setpatch*, *FF*, *Assign*, *bookit*, and finally *MyTitle*. This involves perhaps ten seconds of black screen before *MyTitle* has a chance to run its title music and display its title screen.

The following order of events reduces the black screen considerably:

```
setpatch >NIL: r     ; screen shows trademark
bookit bv            ; screen shows trademark
FF >NIL: -0          ; screen shows trademark
Assign MyVol: CD0:   ; screen shows trademark
rmtm                 ; trademark fades
MyTitle              ; black screen until titles appear
```

Your title may require a substantial load time, particularly if it involves invoking an authoring system which must set itself up before it begins to run your series of commands.

The black screen can be eliminated almost entirely by using the *keeper* program. *Keeper* will display an IFF picture of your choice (probably your logo) until another View is loaded, or it it signaled from within your program, or it is launched a second time with the parameter QUIT. The best position for the use of *keeper* depends on how long your title needs for its initialization, and how tolerant it is to another View occupying display RAM. It *must* follow the *rmtm* command.

> *Games Unkeep The Keeper.* There is an implicit **LoadView** whenever a program opens a screen and displays it. The only exception will be some games programs that address

the display hardware directly. Such programs will remove the *keeper* picture before they start.

A title that requires as much Chip memory as it can sensibly expect to have will load with the following example startup-sequence:

```
setpatch >NIL:
bookit bv
FF >NIL: -0
rmtm
keeper CD0:logo/MyLogo.ILBM
Assign MyVol: CD0:
MyTitle
```

As soon as MyTitle opens a display, the **keeper** task will terminate, close its **View**, and hand back to *MyTitle*. To get the maximum screen memory it can, *MyTitle* will, during its initialization, include the following code, which assumes a procedure called **GetDisplayRam()**.

```
/* various initializations */
task = FindTask("PicKeeper");
/* assuming task is found */
Signal(task,SIGBREAKF_CTRL_C);
GetDisplayRam();
/* load and display titles etc */
```

If an authoring environment is used it will be safer to use the following outline startup-sequence.

```
setpatch >NIL: r
bookit bv
rmtm
keeper CD0:logo/MyLogo.ILBM
FF >NIL: -0
Assign MyVol: CD0:
keeper QUIT
MyTitle
```

The black screen time here will be precisely the time needed for the title to boot and open its first screen. If you are certain that your title leaves enough memory free for the *keeper* **View** and can still start properly, simply omit the second call to *keeper*.

There is no simple guide for startup-sequences except the need to organize events in such a way that all initializations can be done without the user wondering if the system is dead.

Editing the startup-sequence is usually the last stage before pre-mastering a CDTV title. It has some unpleasant possibilities to it. For example, a tiny slip in an *Assign* statement can produce an expensive and useless disc. It should therefore be thought about in advance of the pre-mastering, use the minimum set of commands possible, and interleave these with the *rmtm* and *keeper* programs to produce the least user irritation. Some things will have to be guesswork. When running a title on hard disk, the *bookit* program cannot be used. When a CDTV is booted from a floppy drive, the trademark sign does not appear. It is a good idea to design the startup-sequence before making the first Check Disc, even if this is intended to be used with a floppy or SCSI drive for further development. That way, you can be reasonably sure that the loads occur in the right order and with sensible timings.

The Preferences settings in devs:system-configuration are just as important as using *bookit*. It is always safest to have completely null Prefs fields. For example, *bookit* will remove the pointer during its life, but it will reappear as soon as the main program starts. Blanking the pointer should not, therefore, be left to *bookit*.

Distracting The User During Loading

There will be moments when data must be sought and loaded from the CD. The users may reasonably expect that they will get some clues about what is happening. Expecting them to watch the front panel light is not an adequate feedback. The traditional computer world solutions to this problem include:

- Nothing.
- "Busy" icon.
- "Loading—please wait".
- Writing dots to the display periodically.
- Graphic bar that fills as the load progresses.

Most CDTV titles will be multilingual and avoid any language specific screen prompts.

"Busy" signals and sound jingles are the easiest solution. Fancy animations to cover loads may take longer to load and activate than the load time for the data they are masking. Jingles that occur frequently can be irritating. It must be obvious to the most naive users that they are waiting for something to arrive.

A good "busy" signal screen will have some animation, if only color-cycling, and indicate that a request is being satisfied. Shopping may provide a good analogy. You order a CDTV from a shop assistant who vanishes and *stays* vanished; you wonder if you have been forgotten and eventually, you leave the store and buy the CDTV somewhere else. In a good store, the assistant will telephone the order to the stock room and then distract you by showing you a whole range of CDTV titles you may wish to buy. At least the assistant will say "This will take about five minutes. Why don't you look around and I'll tell you when your CDTV is ready for collection."

CDTV does not have a standard "busy" iconography like the "Zzzzzz" on Amiga Workbench 1.3 or the hourglass on Windows™. Good equivalents on CDTV are spinning CDs, graphics of movement along rows of books on a bookshelf, elevators rising through several floors of a building. Since they will be used often, they will be stored in RAM as *sbox* items and unpacked with the *debox.library* functions when needed. In complex titles they will be context dependent, and show that a map is being found, or a novel, or a musical composition.

The target should always be to minimize load times and inform users that their choice is understood and is in progress.

Directory Structure And Disc Geometry

CDTV uses an international standard for directory and data arrangement known as ISO-9660. Because the CDFS understands this system, it is *not necessary* for programmers to worry about it; very few titles will address the ISO-9660 structures directly, but will rely upon the CDFS to handle this.

For optimization of CDTV performance it is valuable to understand the basics of ISO-9660. The ISO-9660 filing system is very simple. It is designed for a *read-only* medium and need not involve the complexity required for file writing and update. This simplicity may be harnessed to achieve the best possible performance from the medium.

Normally all files will consist of contiguous blocks of data. Each block is a logical sector of 2,048 bytes. (*Other sector sizes are allowed, but the current buildtrack software uses 2048 byte sectors*) Files always start on sector boundaries, so that a file's starting position from the beginning of the ROM track will always be `modulo 2048=0`. *The ISO-9660 standard does allow for file interleaving, but this will be a very unusual state on a CDTV title.*

The root directory contains pointers to files at a logical block. A file may contain data or a directory. The system is hierarchical, and permits directory nesting to a depth of eight. (This restriction does not apply to the CDFS, which has extensions to allow many levels of directory.) In addition to the root directory block there are "path tables". Path tables allow quick access to directories, and are usually held in memory by the CDFS. It follows that the most efficient file access under CDFS happens when the path table can be used. This has a consequence that may appear strange.

> *Many Small Directories Beat One Large Directory.* It is more efficient to use many directories containing a small number of files than one directory containing many files.

Data should be organized to ensure that the laser has to move the smallest possible amount between seeks.

The system performs best on single files, so that a complex file which contains all data (sound, pictures, notes etc.) may be more efficient than a group of files containing separate component elements.

In practice, only file transfers that require the highest level of optimization (e.g., CDXL files) will be forced to cluster their data inside one file. Problems arise when reading from multiple files when file A is at the hub of the disc and file B is at the periphery, because the laser may take up to 0.8 seconds to make its physical move between the inner and outer edge of 600 MByte disc.

The performance of a title on hard disk is not a reliable guide to its performance on a CD-ROM. The organization of data under the Amiga Fast Filing System (FFS) on a hard disk is radically different from the organization of data on a CDTV CD-ROM. Generally the hard disk will be much faster, but this may not be the case with heavily fragmented files on large drives with slow seek times. One of the advantages of the ISO-9660/CDFS system is its regularity and predictability. There is *no* fragmentation.

A good question, then, is *How do I lay out my directories for best performance?*

The *iso* software on the Iso_Devpac disk creates a text file called "controlfile". This text file is used by the *buildtrack* program to create the ISO-9660 directory structure, path tables, and data blocks exactly as they will be arranged on the CD-ROM. The controlfile can be edited with a text editor. The rules are simple: directories can be rearranged, file order can be changed, but files may *not* be moved out of their directories. The *iso* program will pick up the file order from the AmigaDOS file system on the hard disk. Since it is not possible to reorder directories on working hard disks, the order may be arbitrary, or alphabetically sorted, or sorted by size (the default for files in a directory).

If we assume that directory Alpha contains data closely coupled to directory Theta, it is desirable on the CD-ROM to have the data files in these directories as close together as possible. Note the distinction here—the directories themselves are merely system pointers to data segments of one or more sectors on the CD-ROM. However, *buildtrack* will allocate data segments to the files in the order it is presented with the files. By keeping the directory's file count low and positioning closely-coupled directories together in the controlfile you can ensure that the data will be reasonably

close on the CD-ROM, and so eliminate unwanted laser moves between file accesses. You can also change the order of files within a directory to minimize head moves between file accesses. The relatively slow seek time of a CD dictates that getting into a "disc thrashing" situation is something to avoid at all costs.

Compared to the working methods used on hard disks, this has some evident advantages, i.e., almost everything is under your control. The cost is a little extra effort after the data gathering and programming are finished. Attention shifts to disc layout and optimization, and significant time should be allowed for this in project planning.

A number of strategies exist for ensuring that the data sets are as close as possible.

Buildtrack

The "buildtrack" software can make an ISO-9660 image into an AmigaDOS file. This file can then be examined with the *isodir* program. *Isodir* provides a full listing of each directory and file's position in the CD image. Where the program picks up a group of files in close association, by merely checking the start sector for those files it is possible to calculate approximately how long it will take to seek to each file before transfer of data can begin. We may discover, for example, that a gap of 200 MBytes exists between file A and file B because there are several large directories in between. This gap will cause a significant delay, and if file C lies close to file A, then the traverse will happen a second time.

Naming Conventions

A second, less labor intensive technique is to apply naming conventions for directories and files to ensure that an alphabetic sort will place the components in the best order.

Match Data Organization To Data Presentation.

An optimum CD geometry exists for any title, but this may be impossible to determine *a priori* if thousands of branching decisions are make by the user. Normally, it will be possible to assign priorities to data sets. For example, if a title is making a major change of focus, as for example, when an Atlas program changes country or continent, the user may tolerate some waiting; it is intuitive to think of going from one continent to another as involving delay. If that title takes the same amount of time to go between adjacent streets the user will probably be irritated. This is a case where keeping all mapping information in one directory, all text in another, all sound in a third etc., is not helpful. It is better to break the directory structure down by continent, country, and town, ensuring that the big laser moves happen where the granularity of the data is largest. This requires no significant adjustments of the directory order on the CD. When fine tuning the title, it may simply be desirable to change some file ordering, but even that is less likely.

Speed Up by Duplicating

The perceived speed of a title may also benefit greatly if frequently needed menus, pictures, or sound data files are duplicated in such a way that the current directory contains a copy. If all the menus are neatly tucked away in a directory called "menus", this is a traditional, sensible and logical way of ordering data, but it may cause unwanted tracking backwards and forwards.

Keep Directories Small

For highest efficiency in the CDFS buffers directories, the number of files in one directory should be restricted to forty.

Using CDXL To Load Data

Let's say that at a certain point in a title's progress on CDTV, it requires the following data to be loaded into memory:

- 96,000 bytes of sound data.
- 2 pictures—64,032 bytes and 128,016 bytes.
- 938 bytes of text data.

This will require four calls to the AmigaDOS **Open()**,**Read()** and **Close()** functions, together with error checking. Despite the efficiency of the CDFS, there will be a noticeable break between each load. If the component data objects are widely separated on the disc, there may also be some unwanted seeks involved.

If, at the simplest level, these files were to be concatenated with the *Join* command, the data would be contiguous and would require only one **Open()** and **Close()** for access. However, it would still require four **Read()** calls to the various buffer positions.

The CDXL commands to *cdtv.device* allow a high speed DMA transfer of this data. (For a full explanation of CDXL, refer to the "CDXL Overview" article in the CDTV Specifics subsection.) The price for this gain in transfer efficiency is extra effort in design. It will be necessary to write "software tools" to arrange the mixed data into a file and include the necessary information about the size, order and nature of the data chunks within this structure.

The order in which the CDXL transfer list is constructed depends on what needs first to be displayed. For example, assuming that the 640 x 200 x 4 picture is needed on the screen as quickly as possible, it is placed first in the transfer list. The **DoneCode** callback item in the transfer list item for that picture will not display the picture itself, but will signal the main process that the data has arrived. The CDXL transfer will continue with no interruption:

```
void PicReady()
{
Signal(MyTaskPtr,MySigBit);
}
```

Having established a way of finding out when the picture is ready, the main process can initialize the CDXL transfer of the data and do other things until the picture actually arrives:

```
SendIO(MyIoRequest);          /* send CD_READXL command */

/* additional actions as required */
sigs=Wait(1<<MySigBit);       /* picture ready when Wait completes */

LoadRGB4(&MyViewPort,&BufferColorTable,NumColors);
LoadView(&MyView);
/* picture now on screen and transfer still running */
```

This is the fastest way to get complex data from the CD-ROM into memory. Given that multimedia applications often have associated sound, pictures, and other material, it is well worth the extra effort (which is not great) in setting up the linked transfer list. The major effort involved is in getting the associated data into one file, or if not into one file, into a series of contiguous sectors on the CD-ROM. This technique absolutely depends on the data elements being sequential.

A more complex extension of this approach to data management is to slice up sound and picture files and load a series of pictures transparently while a continuous stream of sound is being spooled

from the CD. This demands a very tight control of data sizes, sound playback frequency and other timing considerations.

As a general principle, optimization of CDTV titles frequently depends on shifting the burden of computation from run-time to the development stage.

Using Burst Mode

Burst mode is not a special function that you can switch in or out, and the term can give rise to confusion. The CD mechanism and circuitry buffers sector transfers internally before sending the data in a "burst" onto the data bus. This happens transparently. The consequence of it is that large continuous transfers of data are more efficient than many short transfers. A second consequence is that data transfer is not a linear time function.

Avoiding ILBMs

The IFF ILBM file format for pictures has been very successful in promoting easy exchange of pictures between application programs. The first line from each of the picture's bitplanes is run length encoded, and then the second line and so on until the whole image has been converted. The advantage of this scheme is that if a picture is decoded from a slow storage medium, each successive line can at least appear with all its color and information intact.

Unfortunately on an unexpanded CDTV, the ILBM format requires the 68000 to parse the IFF file, unpack the rows, reorder the color map and do various housekeeping tasks. The 68000 is running in Chip memory, and is losing cycles to the custom chips and DMA. The result is a distinct delay before the picture appears.

An older IFF picture format, the ACBM (Amiga Contiguous BitMap) provides a partial solution. In an ACBM, the BODY chunk is replaced by an ABIT chunk which contains the data for each bitplane in flat format. An Amiga Basic program called *SaveACMB* will convert ILBMs to ACBMs. Developers using authoring systems should check whether their authoring system will read ACBMs. If it will, substantial speed increases can be gained by using this format.

Developers who are coding titles themselves may want to consider whether they should bother with the IFF format at all once the graphics work has been completed for pictures other than ANIMs. If the title uses an arbitrary mixture of picture formats, it will be difficult to dispense with the header information, but the more commonality that is imposed on all the data types used, the greater the potential optimization available.

Generally, Intuition will not be used in CDTV titles because it is not designed for viewing and operating from a distance of ten feet on a television screen. Programmers are more likely to create their own custom **Views**, **ViewPorts** and **RastPorts** and render directly to these. From here it is a small step to an arrangement of bitplane and color information that allows a single DMA transfer of the data with no interruptions and no parsing.

Contiguous bitplane date for picture	Color Information	8 byte spare RAM (extra bytes to avoid DMAC problems)

If the file format adopted conforms exactly to the layout shown above, the fastest possible load time is obtained. This scheme relies on programmers taking control of their own graphics structures. If memory requirements are tight, it is desirable in any case to manage an internal heap for picture and sound buffers, with a variety of Views managing different display ratios.

For example, if 327,744 bytes of Chip memory are allocated at start up, the space will allow for two 640 x 512 eight color picture buffers to be available at any time, or five 320 x 256 HAM pictures. The load-and-display routines can be organized to give double-buffered displays over a wide range of picture formats with very little processor use and no possibility of out-of-memory problems. Unless a CDXL scheme is used for loading pictures, the normal scheme becomes reversed.

Normal

- Load Header.
- Examine Header.
- Set Up Screens And Windows.
- Move BitPlane Data Into Buffer.
- Apply Color Mapping.

Optimized

- Select Appropriate Buffer.
- Load All Data.
- Use ViewModes To Determine **RastPort & ViewPort.**
- Load Colors.
- **LoadView()**.

The price for this is that the picture format is no longer compatible with paint programs and changes may be difficult to make. So when designing software tools to take standard IFF format files and transform them into optimized objects, it is worth the small extra effort required to make this a two-way process.

Ideally the graphics engine for the title will be capable of loading all formats, and a very late stage in development will "batch process" the picture files into the optimized layout.

Asynchronous Reads And Multiple Reads

Where possible, the multitasking power of the Amiga/CDTV system should be used to reduce *perceived* loading times. One obvious example of this is asynchronous loading of data. While something wonderful is happening on the screen, the next picture or sound is invisibly moving into memory ready for rapid display. This can be achieved at three main levels:

- Process
- Task
- Device

The advantage of a process over a task is that a process can communicate with DOS and a task cannot. Processes may be spawned from inside the main program, and talked to with signals, semaphores, or messages. Alternatively, a process may be a separate program that is *Run* from the startup-sequence, has a named **Port**, and hence may be sent messages from several multitasking programs that constitute the title. This approach will be successful for relatively simple cases such as audio management, but may become very complex for control of pictures.

Tasks are simpler (though not much simpler) to launch from within a program. Because they are "locked out" from DOS, however, they lose the simplicity of being able to find and load a file. Tasks can use the system devices—*audio.device*, et al.

The power of CDTV resides on the strength of the *cdtv.device*. The high-level facilities of the DOS interface rely on low-level commands to *cdtv.device*. The *cdtv.device* is not a filing system, and consequently does not respond to file names or other file attributes. It requires commands to transfer data at a logical rather than symbolic level.

The decision that programmers make about when to use a process, a task or a device call depends on the degree of direct control they require, or the degree of binding between the main program and the loader code. The use of processes and tasks provides generality, but requires very careful control of task/process priority. Particularly under Kickstart 1.3, it is possible to create circumstances where a separately launched task will *never* run. There may be circumstances at run-time when different priorities apply. "Race conditions" can arise where the order of completion of operations can be very complex under different loadings.

Using processes and tasks for asynchronous data handling is appropriate in a situation where, for example, an authoring system allows external calls via ARexx but does not provide the facility to handle large stereo sound files. It is unlikely to be the ideal solution in a complex multimedia program which is seeking the highest possible control, because the order of operations has been handed over by the programmer to the Exec task switching mechanism.

Asynchronous data loading with the *cdtv.device* is a simple matter provided that the location of the file on the CD-ROM data track and its size, is known or can be determined. These two items of information are available through the CDFS.

The first sector of the file is obtained by calling the DOS **Lock()** function with a pointer to the name and the access mode required, which is certain to be ACCESS_READ on a CD! Assuming the file exists, **Lock()** returns a BCPL pointer to a lock. After conversion to 680x0 pointer, the**Lock** is in fact pointing to a **FileLock** structure in memory. The second field of the structure is **fl_Key**. This is a LONG (32 bits) giving the sector where the file starts. No further information is needed to locate the data because the file will not be fragmented.

The size of the file can be obtained by passing the original BPTR to the DOS function **Examine()**, which also requires a pointer to a **FileInfoBlock**. The **FileInfoBlock** structure field **fib_Size** contains the file size in bytes.

> *A Global Structure Makes A World Of Difference.* Obtaining the location and size of a file by the method described above cannot be performed from a task, because it requires DOS access. If your asynchronous loader is a sub-task, it should therefore be allowed to access a global structure in which the main process places the location and size before signaling the task, or be sent a **Message** which includes the information.

If this file will be needed frequently, it may be useful to store the position and size values, and in many instances where programmer-defined data structures of fixed size are involved there may be no need to call **Examine()** to determine the size.

With the information to hand, one command to *cdtv.device* will load the file, or part of the file.

Assuming an **IOStdReq** structure has been allocated, initialized, and opened successfully on *cdtv.device*, the following example performs a fast asynchronous load:

```
#define SECTOR_SIZE 2048;

        MyIOReq->Offset  = StartSector*SECTOR_SIZE;
        MyIOReq->Length  = FileLength+8;
        MyIOReq->Data    = &MyBuffer;
        MyIOReq->Command = CD_READ;
        SendIO(&MyIOReq);
```

Notice that SECTOR_SIZE may (rarely) be different on different CD-ROMs, but will currently be 2,048 bytes for CDTV discs pre-mastered with existing utilities. The extra eight bytes added to the Read length is vital for direct transfers, and in this instance MyBuffer needs to be eight bytes longer than its significant size. This is because the DMAC system may very occasionally (1% chance) corrupt the last eight bytes of data transferred.

While the **SendIO()** is pending, other operations, including preparing and sending further requests to the *cdtv.device*, may continue—monitoring user I/O, screen painting or playing memory resident audio samples. Periodic calls to **CheckIO()** will report the status of this I/O request, and finally **WaitIO()** will put your program to sleep until the operation is complete.

Given a handful of **IOStdReq** structures, a series of such commands may be queued to the device. However, where the number of queued items or the complexity of the data becomes significant, it may well be easier to set up a CDXL transfer list. But for most load operations, it will be significantly faster to use the CD_READ command than to rely on CDFS which has to maintain a broad accessibility.

Unfortunately, *cdtv.device* I/O will not work on a hard disk. It requires the presence of a CD-ROM, and so usually will be brought into operation once the data are finalized and available on a Check Disc CD-ROM.

Optimization Summary

There is remarkably little that can be done to optimize titles created with authoring systems that do not include the relevant low-level facilities, except scrupulous attention to directory layout and disc geometry.

For those not using authoring systems, the following summarizes what has been discussed.

- All titles, using whatever optimization, are vulnerable to poor layout.
- Custom file formats and internal memory organization can create a system in which data loading can be optimized.
- Asynchronous loading of data which is known to be required next can eliminate many delays.
- The *cdtv.device* provides all the facilities needed to ensure that the flow of data into memory is efficient in the foreground and possible in the background.
- Where possible, computation should be performed "off-line" during development.
- Careful thought given to data formats can simplify programs and radically improve performance. This is a data-driven system.

Optimal Disc Layout

A primary key to improving the performance of a CD-ROM-based application is to eliminate as many seeks as possible. While this can be done by economizing the title's disc accesses, many further gains may be realized by thoughtful layout of the files on disc. With some knowledge of the ISO filesystem structure and taking advantage of the CDTV Device File System 's (CDFS) caching, significant speed gains can be achieved, improving title performance.

ISO Structure

The structure of the ISO-9660 filesystem is covered in more detail in the "CDTV File System" chapter. However, only some basic facts are needed to understand the techniques that follow.

ISO-9660 requires that files occupy contiguous sectors on the disc, permitting the entire contents of the file to be scooped up in a single operation. Because of this, there are no "side-sectors" as there are with AmigaDOS. Thus, there is no secondary seeking to collect block lists. The filesystem can go straight from the directory entry to the file data.

ISO directories can also be thought of as a special form of file. Unlike AmigaDOS, which reserves an entire disk block for each file descriptor, ISO packs together as many descriptors as will fit in a single block. Directories that consume more than one block are stored, like files, in contiguous blocks. In this way, an entire directory can be read all at once.

In order for a file to be found, the directory that contains it must be scanned. Directories themselves, however, can be found by scanning the path table, which is loaded by CDFS upon disc insertion.

Cache Operation

CDFS has a rudimentary block caching system. Two sets of caches are maintained: one for data blocks, and one for directory blocks. The sizes of both caches can be set though the Boot Options (covered in the "CDTV File System" chapter).

The data block cache is a simple read-ahead cache. If a requested block is not in the cache, the entire cache is dumped and refilled with the requested block and the *N* blocks following it. Thus, the read-ahead cache's greatest value is realized by doing sequential reads.

The directory block cache is a bit more intelligent. When a directory is loaded, its blocks remain in the cache until the cache is full. If filled when a free cache entry is required, the least recently used entry is dumped and recycled. When a new directory block needs to be read, it is requested through the above-described data cache. Thus, a directory scan can cause the data cache to be dumped. Note however that, if directories are grouped together, the data cache gets dumped only once. Any further directory block requests that immediately follow are satisfied from the data cache.

The cache operation is described here only to give you the most general of ideas as to how organize your discs. The actual internal workings of CDFS are private. Commodore reserves the right to alter the caching mechanism in the future to improve performance.

Analyzing Your Title

Before you can knowledgeably lay out your disc, you must first know which files and directories your title accesses, and the order in which it does so. Some of these file accesses are non-obvious. An example would be the DOS **Execute()** function, which first loads *C:Run* before loading the specified program.

One tool that can help is *PickPacket*, a public domain program that wedges into any DOS device and prints a report of all DOS operations requested of the device. *PickPacket*'s output can be used to more accurately determine your title's sequence of reads and file accesses. You can also use the Commodore-supplied tool *OptCD*, which uses the built-in statistics-gathering features of CDFS V26.

Goals

Finally, before you start laying out files, you should decide what you're trying to accomplish. Obviously, you're trying to optimize disc accesses, but optimizing for one case may leave other cases in very poor shape.

While collecting the raw data, use the title yourself (or better, have a disinterested party do it) and determine which parts seem "slow". In particular, ask these questions:

- Does the title start slowly (more than 5 seconds)?
- Does moving from one area to another seem to take a long time?
- If it has a search facility, does it take a long time to find something?
- Is there any part of the title that seems to take longer than it should?

If you feel the title is slow in a specific area, you may wish to pay particular attention to the reported disc activity when the title is operating in the area of interest.

Disc Layout

Once you have an idea of where your disc accesses are occurring, and which ones you wish to optimize, you can start laying out files. Though you can't specify a particular block number when creating an ISO image, the mastering software will let you specify the order in which files and directories are laid out, giving you a high degree of control over file grouping.

While no single approach will yield the ultimate optimized disc, some general guidelines will yield good results.

The startup-sequence can be the biggest offender in your boot times. Group all the commands that appear in the startup-sequence together on the disc in the order in which they appear in the startup-sequence, and put the startup-sequence itself immediately before all of them. This will cause the commands to be pre-loaded into the read-ahead cache when the startup-sequence is read. Keeping your startup-sequence short also helps a great deal.

Group small files together (8K or less), and group them in the order in which they're accessed. In this way, when the first such file is opened, the rest will be pre-loaded by the read-ahead cache.

Also, consider recoding your application such that all the small files are accessed at once, rather than interspersed with accesses to large files. You may even wish to consider caching all such files in RAM, thus requiring them to be read but once.

Open all disk-based shared libraries at once, and group them on the disc together in the order in which they're opened.

Don't be afraid to have large directories on an ISO disc. Recall that ISO directory entries are packed together; thus, more than one entry can be stored in a disc block, and these blocks get cached by filesystem. A large directory, if accessed frequently, will remain cached. Opening files happens more quickly from a cached directory.

You may care to experiment with leaving system files (libraries, devices, fonts, etc.) in the root directory. At boot time, if DOS can't find, for example, a directory named "libs" in the root of the boot volume, it will assign LIBS: to the root, and all libraries will be searched for there. By doing this, you can keep the system from consuming directory cache entries for "libs," "devs," "fonts," "l," and "c." Note that there is a tradeoff here between more efficient cache usage and a really cluttered root directory. You'll have to weigh both sides and decide accordingly.

The Startup-Sequence *must* be located in a directory called "s", or the system won't find it.

If your title will have Workbench icons on it, we strongly recommend you group all the .info files together, and lay them out in alphanumeric order. ISO mandates that directory entries be sorted in ascending ASCII order, which means the DOS function **ExNext()** will encounter them in that sequence. Therefore, when Workbench starts searching a directory for icons, it will see them in alphanumeric order. If they are laid out in this way on the disc, all the icons will be loaded in a single contiguous read, and they will appear on screen *very* quickly.

If you have files which are seldom accessed, you might place them in a separate directory, and locate those files and the directory away from the more-frequently accessed files and directories. This way, you won't have a "speed bump" which needs to be skipped in the middle of your critical files. Keep in mind, however, that doing this can make these files harder to access, and could make operations involving these files slower than they were before.

If your title uses CDXL, you might also consider moving all the CDXL files in a separate directory and isolating them, too. Since CDXL doesn't go through the filesystem, it's to your advantage to have CDXL files located away from "normal" files.

You might even consider dispensing with the filesystem entirely and using CDXL for all your data-loading needs. It's extremely flexible and blindingly quick. Search your title for file operations that could be done using CDXL instead.

Disc Access

While disc layout is important, so also is the manner in which your title accesses the disc.

Make your read lengths as long as possible. If your application reads data in very tiny chunks (less than 1K or so each), the most efficient use of the system is not realized. The filesystem and the CD-ROM prefer to deliver the largest number of bytes possible. If your application requires piecemeal reads, consider incorporating buffered I/O routines. Most C compiler packages include them with their link libraries. Set your buffer sizes to 2K or more.

Try to minimize "file hopping" (switching frequently between two or more files) as it is seek-intensive. Try to process individual files in their entirety.